On the Knowledge-based Synthesis of Data Structure Manipulating Programs (Mathematical Methods in Software Science and Engineering : Second Conference)

AUTHOR(S):

MANO, NOBUOKI

数理解析研究所講究録
396巻 1980年 236-259

236

ON THE KNOWLEDGE-BASED SYNTHESIS OF DATA STRUCTURE MANIPULATING PROGRAMS

Nobuoki Mano

Computer Science Division
Electrotechnical Laboratory

1-1-4,Umezono,Sakura-mura,
Niihari-gun,Ibaraki-ken,305
JAPAN

ABSTRACT

Synthesis of data structure manipulating programs is described. The method proposed in this paper synthesizes plans, which represent logical structures of programs, from user problem specifications -- input-output behavioral specifications , structure description of data and the specifications on the way how the computations are done. The plan synthesis process proceeds deductively, following the structure definitions of data and guided by the specific control strategies dependent on functional categories of problems. In this paper, first, we define elements of program and data descriptions and show related programming knowledge, and then, give the method of plan synthesis and modification with some examples.

1. Introduction

Data structure manipulation programs play the central role in the system program and algorithm study in computer science ([1]). So the synthesis of those programs manipulating abstract and concrete data types, e.g., recursive data structures such as lists and trees, arrays, character strings, hash tables, stacks and compound data structures are very important and interesting themes to be achieved.

Unfortunately, deductive program-synthesis method proposed so far have been of only academic value, not only because their representation of programs lack in flexibility and descriptiveness of important information concerned with programs, but also because their inference mechanisms come from mathematical logic with the uniform treatment of all predicates and it is also difficult to apply programming knowledge for their problem solving processes so that combinatorial explosion of candidate clauses arise.

Recently the design and writing of programs based on the structure of data has been proposed ([4]). We apply this idea and also make full use of programming knowledge to the plan synthesis of programs ([6],[7],[8]). Our problem-solving method in the plan synthesis process is a kind of natural deduction ([3]), based on the structure description of data and guided by the specific control strategies dependent on the functional category of a problem. The main features of this synthesis method are as follows.

1) Plan representation of programs

A plan ([10]) is an internal logical structure representation of program (or a part of it) which consists of an enclosing segment (which represent a problem or a subproblem) and its subsegments and axioms (or lemmas). Here a segment represents a unit of action or test with its input-output specification. A plan include not only dataflow and control-flow relationship between these segments but also logical dependencies between the specifications of the segments and clauses of axioms.

2) Plan synthesis based on the structure description of data

Structure description of data can be effectively used to get the the access path and alternate occurrence, so the logical structure of plan is synthesized by making the case analysis of condition test and dataflow correspond with the structure description of data. Plans for hierarchical data structures and modified plans in the case of programs with side-effects are also synthesized more easily than in any other program-synthesis methods.

3) Plan synthesis guided by functional category of problems

Quantitative relationship between input and output of a problem is the most effective information which decides the functional category of the problem. Main subsegments (or a subproblem segment) which play important role in plans are introduced by the knowledge about the problem category.

We consider those plans consisting only of sequence, conditional test and recursion in this paper.

2. The Elements of Program Description and Programming Knowledge

2.1 Data Types and Segments

The most basic elements of program descriptions ([10]) are the definitions of structured data types, their parts and the relations between them,and the input-output specifications of the behavior of program segments (see Table 1).

1) Data types, their structures and relations

Data types are defined as in Table 1. Their particular objects are represented by concatenating the data type with an identifier number. Recursive data structures are defined recursively by Cartesian product (CP) and disjoint union (DU) functions ([2],[7],[8]). The argument order of CP and DU is significant for plan synthesis and code construction. Relations are defined by ASSOCIATION. Associations and attributes are either primitive or non-primitive. The meaning of non-primitive one is defined by an axiom.

2) Segments and axioms

Segments are functional units - modules or some part of it. Their behaviors are represented by their specifications - input objects, output objects, preconditions and postconditions. Test segments have an output specification for each case. Primitive segments have LISP metacodes, when target programming language is LISP. Subproblem segments are allowed to represent subproblems to be solved. Subproblem segments are introduced by the following two reasons.

(1) There exist no adequate subsegments in the knowledge base.

This occurs usually when the input data of the subproblem segment is set input. The specification of the subproblem segment is synthesized during the problem solving process.

(2) Given a problem, system assigns a subproblem segment and its input-output specification for each structure description. The specification of subproblem segment is synthesized during the problem solving process.

Axioms (or lemmas) are indispensable elements for the synthesis (or verifications of plans). The form of an axiom corresponds with that of the rule in [5] as follows.

< consequent > => < subgoals > [if < condition >]

Condition part is matched with the postcondition of the corresponding case in a test subsegment, or abbreviated.

2.2 Knowledge Structure

The following are the main structures in our knowledge base.

1) Datatypes, and their related associations, attributes and types. ISA (generalization-specialization) relationship between datatypes.
2) ISA relationship between segments and axioms.
3) ISA relationship between associations, attributes and types.
4) Index structure on all the tuple predicate information in the system which point where the tuple instances exist in the pre- and post- conditions of segment and axiom clauses. Information about the tuple (e.g., whether it is an association, an attribute or a type, and whether it is primitive or non-primitive) is attached there.

Some knowledge structures are shown in Fig.1.

2.3 Program Plans

Plans are the logical structure representation of programs which include not only dataflow and control-flow relationship between an enclosing segment and subsegments of a program, but also logical dependencies between the specifications of the segments and clauses of axioms.

The dataflow, control-flow and logical link in the plan are directed except the self-loop around subproblem segments, as we have no loop representation (except recursion) in the plan. This is very convenient for plan synthesis and modification as will be shown in 3.3.

The merits of plan representation of programs are as follows.

1) Plans represent language-independent, essential (without connective tissues) internal structure of programs.
2) A plan has a two-dimensional, net-like structure so that addition and modification of information is easily handled.
3) It has dataflow as an explicit constituents, so it is additive and can represent parallelism (partial ordering). The input and output objects of segments and the arguments of clauses in axioms might have input (or output) data part-id as their binding values. So dataflow can represent where each part of the structure of data is processed in a plan. Plan modification uses this information effectively. Dataflow representation is enough for value transfer in the function type programs, but

free-variable assignments require subsegments representing the actions. Objects which are outputs by virtue of the fact that they have been subjected to a side-effect continue to use their original values.

4) Segments have their behavioral specifications with them so that their effects on other segments around are easily found by following logical links and control-flow. The postconditions of an enclosing segment are goals to be satisfied ([5],[6],[8]), and the antecedent part of axioms and the preconditions of subsegments are considered to be subgoals. Postconditions of subsegments, preconditions of enclosing segment and the consequent clauses of axioms justify these goals and subgoals. Logical operators '&' and 'or' are realized by the sequential and branched control path of justifying segments, respectively.

5) Plan representation matches with programmer's intuition and is of good correspondence with sentential expressions in natural languages.

2.4 Knowledge about Problems

1) Problem categories ([7],[8])

Problems can be categorized by their functions.

* Enumeration of a data structure such as a list or a tree is the most fundamental operation (enumeration is sometimes called traverse).
* Search is the basis of insertion and deletion operation in the structure changing programs.

* Many problems belong to set manipulation problems ([1]). Among them are the set union problems (e.g. insert, merge, append and so on) and set preserving problems (1-input-1-output and the members of output set are equal to those of input set, e.g. sort,reverse, and so on). In these problems, when represented by function-type programs, construction operations are used. When represented by programs with side-effects, set preserving problems use such techniques as swapping of array elements (e.g., bubble-sort, exchange-sort), and pointer-rotattion technique (e.g. list-reverse, Schorr-Waite graph marking algorithm).

2) Computation strategies ([7],[8])

Usually we have plural representation programs for one problem. So we must decide which computation strategy to choose.

* If both a function-type program and a program with side-effects are possible, select either.
* Structure description of data implies a decomposition strategy (e.g. tail-recursion), so select one description.
* If a problem has two set input, select either one-input-decomposition or simultaneous decomposition of both inputs.

Knowledge about problems are embedded in the rules of production systems in our system.

3. Problem-Solving Method

3.1 Problem Specifications

The following specifications are given by the user.

1) Input-output specifications of a problem to be solved.
They are the same format with the segment definitions in 2.1.
2) Definition of input (and output) data structure
3) Selection of computation strategies.

3.2 Outline of the Overall Problem-Solving Process

In most cases, a problem has some problems. We suppose in this paper that subproblems are introduced by the two reasons explained in 2.1.

(1) When a subproblem segment is introduced during the synthesis process

Its input-output specification is determined durig problem solving process of its immediate upper-level plan. The subproblem must be solved next. When the solving process of the subproblem fail-ended, system control backtracks to its parent problem. So a main problem is not said to have been solved until all its subproblem have been solved successfully, if it has them at all.

(2) When the data structure is hierarchical

System analyzes the structure description of data and assigns a subproblem for each description by designing (hypothesizing) the input-output specifications of each subproblem. Then our system tries to solve each problem. If at least one problem fail-ended,

redesign must be done.

The basic cycle of the problem-solving process in one problem can be informally described by the following steps.

(1) Push the exit-entrance pair (or case-exit-entrance pairs) into task-stack.

(2) If task-stack is empty, exit with success.

(3) Pop the task-stack and solve the taken-out pair. If succeeded, go to 2. If case analysis occur, push those pairs between axioms and a test segment into the task-stack, then go to 3. If inconsistency is found, drive the failure back-up mechanism, which is implicitly included in the system ([6]).

3.3 Details of Solving one Problem

3.3.1 The Instantiation of Structure Description of Data for the Problem

From the structure definition of data in the problem specification, an instantiated structure description and its each part name is generated.

3.3.2 Introduction of Main Subsegments from Various Knowledge Sources

Plan synthesis can be seen as an assembling operation of various subsegments introduced from various knowledge sources.

* As for recursive data structures, test and selector subsegments are introduced from alternation and decomposition expression, respectively.
* Principal subsegment which plays the central role in the plan is

introduced from the postcondition of the enclosing segment.

* Some supplementary subsegments are introduced by the particular knowledge about data type (e.g., hashing operation) and knowledge about hierarchical data structures with side-effects (e.g., list store operation in the upper-level plan) and so on.
* Subsegments related with a particular data-type in the concrete plan are often introduced through the knowledge structure (see 2.2), based on the functions of subsegments in its abstract plan (e.g., 'arrayfetch' for implementing portion of 'top' stack-operation).

Only definite dataflow can be drawn when a subsegment is introduced, and it is often useful for deciding whether the subsegment is primitive or it represents a subproblem segment.

3.3.3 Complete Plan Synthesis

1) Goal-directed problem-solving

Problem-solving in a plan proceeds bidirectionally ([9]), namely, from the goal (postcondition) side of enclosing segment and from the entrance side of the plan. But basically goals (and subgoals) take the initiative, and make logical connections by links between them and introduced (as in 3.3.2 or newly) subsegments which match the goals. The preconditions of those subsegments must be justified by the postconditions of segments already exist in the plan or the precondition of the enclosing segment (sometimes through axioms). The accumulated postcondition of segments along the flow of data in a plan are useful for deciding segments and axioms to be introduced and the specifications of subproblem segments.

2) Case analysis of condition branches

Condition branches occur in the following cases with the result of nested scope of test segments, the order of which correspond to those of related elements in the structure description.

* When DU expression exist in the structure description of data.
  Candidate axioms are chosen by the goal pattern first, and then they are further selected by the correspondence of their part-relations intheir antecedents with the structure description of data.
* When the relation between two input elements of the enclosing segment is undefined.
  In this case a test segment is introduced from the knowledge about goal information. All possible cases except unnecessary one from the premise of the problem are enumerated and each case has its relation as its assumption.

3) Generation of flow information

Every flow is directed from entrance to exit except self-loop in the plan.

Each data flow is either between a subsegment and its enclosing segment or between two subsegments. In the former case, when clauses like below exist in the subgoal part of an axiom (Fig.2(a)),

$$(\text{rel2 } [\text{rel1 1}]\ \text{n}) \equiv (\text{rel1 1 m})(\text{rel2 m n})$$

and logical link is generated both between AXIOM-C and SEGMENT-A, and between AXIOM-C and SEGMENT-B, a dataflow from the output of SEGMENT-A to the input of SEGMENT-B is drawn. In the latter case,

when a logical link is found and the argument of the subsegment gets a definite value, dataflow between the subsegment and input or output of its enclosing segment is drawn (Fig.2(b)). Fig.3 is the plan of the program to find the maximum and its index of an array a[0:n] ([5]), using subsegments ASSIGN with side-effects. Note the dataflow from MAXINDEX-2 to LESSTHAN-1. Also note the independent dataflow of maximum value and its index. Distinction of dataflow for each case of branch and their join is required.

When there exists case analysis, control-flow is drawn incrementally, following the progress of problem-solving. This control-flow can be used to get the outer information from the inside of a case scope. Drawing control-flow is easy.

4) Simultaneous goals

Quantitative relationship between input and output is the basis of problem- solving, so the goal concerned with set relation is first tried, and it is assumed. Then the other type goal is tried. If the latter goal is found to be inconsistent with the previous assumptions, control backtrack to the first goal. For example, sort problems have simultaneous goals; permutation and sorted ([6],[7]). In this case, simultaneous goals are the cause of introducing subproblem segment 'insert', because of function type program and first-rest decomposition of list. Subproblem 'insert' is solved as described above.

3.3.4 Modification

Fig.3 can be seen seen as a modification example; first, the plan of finding maximum value of the array a[0,n] is synthesized, and then the plan is modified to get the index of the position of

the maximum value. In this case, because of the mutual independence of these goals and the features of plan representation, we need no such complicated modification procedures proposed in [5]. Generally, plan modification can be done rather easily, not only because the effect of modification is easily found as segments have their specifications with them, but also the correspondence between structure description of data and the plan structure are explicitly taken by our system.

3.4 Plan Synthesis of Allall Program ([5])

1) Input-output specification

```
(DEFSPECS  allall
  (INPUT     list-1 list-2)
  (PRECOND   (list list-1) (list list-2))
  (CASE-1    (OUTPUT TRUE)
             (POSTCOND (< list-1 list-2)) )
  (CASE-2    (OUTPUT FALSE)
             (POSTCOND (not (< list-1 list-2))) ) )
```

Here,

```
(forall (object-1) (member list-1 object-1) =>
                     (predicate object-2 object-1) )
```

is abbreviated as

(predicate object-2 list-1) .

2) Structure definition of input data

(DEFMEMBER list-1 integer) (DEFMEMBER list-2 integer)

3) Strategy

(DECOMPOSE list-1)

4) Instantiation of structure description of data

(list integer) = (DU (CP) (CP integer (list integer)))

list-1 int-1 list-3

5) Introduction of main subsegments into the plan (Fig.4(a))

From the structure description of input data, NULLTEST-1 is introduced by the DU expression and input list-1 is decomposed into two parts by the selectors FIRST-1 and REST-1 following the CP expression. The goal of the problem means the judgement on the relationship between two inputs of the problem. So the output of FIRST-1 subsegment (i.e., int-1) is fed to a subsegment whose role is comparison test, but the other input of the subsegment is list so that the subsegment becomes a subproblem segment called LESSTHAN-1. The output of REST-1 subsegment is fed into ALLALL-2 subsegment (recursive call of the enclosing segment) because list-3 is defined as list-1 recursively.

6) Goal directed problem solving

After the goal-directed problem solving has done, the logical links of Allall plan is shown in Fig.4(b).

7) Complete plan

Dataflow and control flow of Allall plan is shown in Fig.4(c).

8) Program

```
allall(l1, l2) = if empty(l1) then true
                 else if lessall(first(l1),l2)
                      then allall(rest(l1),l2)
                           else false
```

9) LESSALL problem can be solved almost same way, but it has no subproblems.

4. Conclusion

In this paper, we have given a program-plan synthesis method

based on the structure description of data and guided by the knowledge on functional category of problems. This method belongs to the knowledge engineering approach and has a human-like synthesis process. The author believes that this synthesis method is very hopeful as it is good at plan synthesis by modification which is required very frequently in practical applications, and that it can contribute to software engineering when developed further.

Acknowledgement

The author is grateful to Drs. Osamu Ishii and Akio Tojo for the oppotunity to make the present study and to the staffs of the Information Systems section and Dr. Torii for their helpful discussions.

References

[1] A.V.Aho, J.E.Hopcroft and J.D.Ullman, "The Design and Analysis of Computer Algorithms," Addison-Wesley (1974)

[2] W.H.Burge, "Recursive Programming Techniques," Addison-Wesley (1975)

[3] A.Hansson and S.Tarnlund, "A Natural Programming Calculus," Proc. of the 6th IJCAI (1979)

[4] J.H.Hughes, "A Formalization and Explication of the Michael Jackson Method of Program Design," Software- Practice and Experience, Vol.9 (1979)

[5] Z.Manna and R.Waldinger, "Synthesis: Dreams = Programs," Stanford Univ. AIM-302 (1977)

[6] N.Mano, "Program Synthesis Based on the Concept of Dataflow," (in Japanese), Institute of Electronics and Communication Engineers of Japan (IECEJ), AL79-70 (1979)

[7] N.Mano, "On the Knowledge-baed Synthesis of Recursive Data Structure Handling Programs," (in Japanese),IECEJ, AL79-120 (1980)

[8] N.Mano, "On the Synthesis of Data Structure Handling Programs," (in Japanese), Information Processing Society of Japan, Preprint of WGAI 15-3 (1980)

[9] N.Nilsson, "A Production System for Automatic Deduction," Stanford Univ., HPP-77-28 (1977)

[10] C.Rich and H.E.Shrobe, "Initial Report on a LISP Programmer's Apprentice," IEEE Trans. on Soft. Eng., Vol.SE-4, No.6 (1978) AIM-302 (1977)

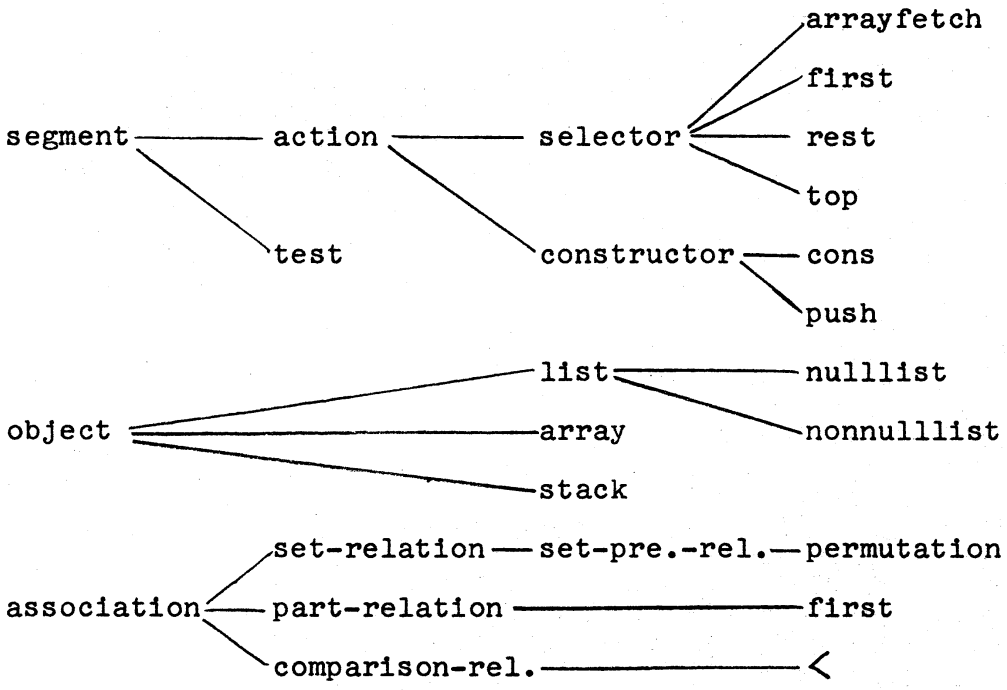
Table 1. Representation of List Concept

```
(DATATYPE list (TYPE PRIMITIVE))
(DEFSTRUCT listdecomp (list object) = (DU (CP) (CP object (list object)))
(ASSOCIATION (first list object) (PREDTYPE PRIMITIVE))
(ASSOCIATION (rest  list list)   (PREDTYPE PRIMITIVE))
(ASSOCIATION (permutation list list) (PREDTYPE NON-PRIMITIVE))
(ATTRIBUTE   (sorted list)  (PREDTYPE NON-PRIMITIVE))
(TYPE  (nulllist list))
(TYPE  (nonnulllist list))
(PREDICATES listdecomp (nulllist nonnulllist))

(DEFSPECS first
  (INPUT    list-1)
  (PRECOND  (nonnulllist list-1))
  (OUTPUT   object-1)
  (POSTCOND (object object-1))
  (METACODE (car list-1)) )
(DEFSPECS nulltest
  (INPUT    list-1)
  (PRECOND  (list list-1))
  (CASE-1  (POSTCOND  (nulllist list-1))
           (METACODE  (null list-1)) )
  (CASE-2  (POSTCOND  (nonnulllist list-1))
           (METACODE  T) ) )
(DEFAXIOM  vacuous-axioml
  (CONSEQUENT (comparison-relation object-1 list-1))
  (CONDITION  (nulllist list-1))
  (SUBGOALS   (boolean  TRUE)) )
(CONSTITUENT-SEGMENTS list (first rest cons nulltest - - ))

(ISA first selector (TYPE SEGMENT))
(ISA nulltest test  (TYPE SEGMENT))
(ISA first part-relation (TYPE ASSOCIATION))
```

Fig.1 Some ISA Structure in the Knowledge Base



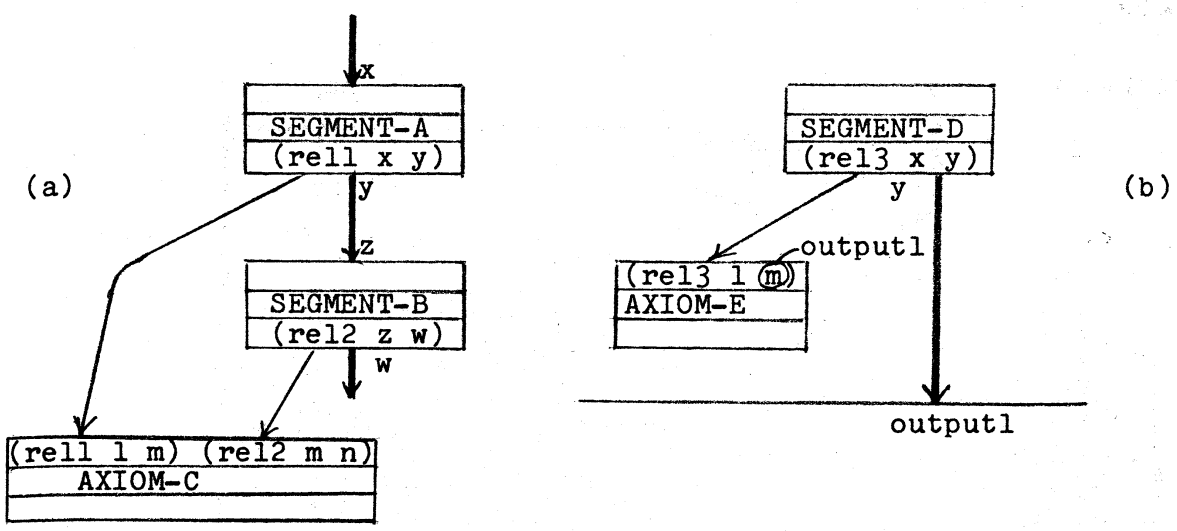
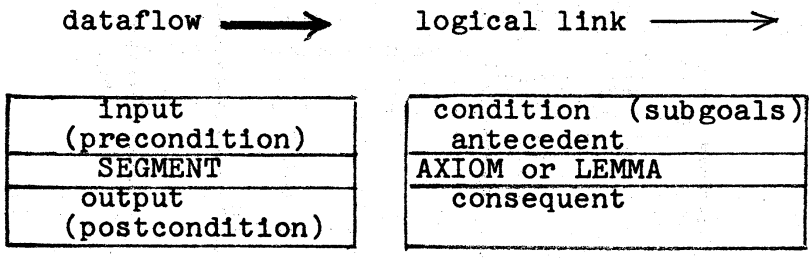
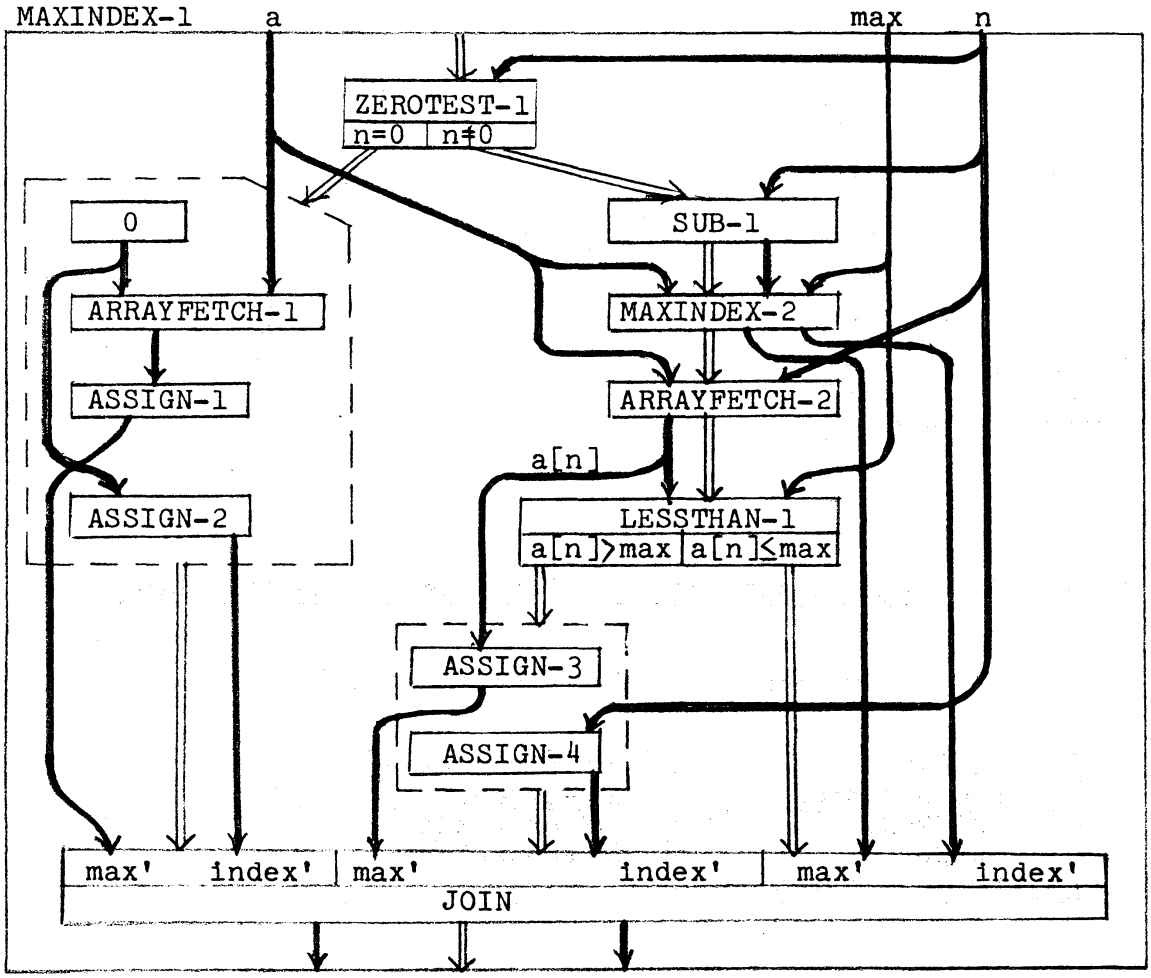


Fig.2 The Relationship Between Dataflow and Logical Link





Fi.3 Plan Representation of the Program to Find the Maximum and its Index of an Array a[0:n]

dataflow ➡ control-flow ⟹

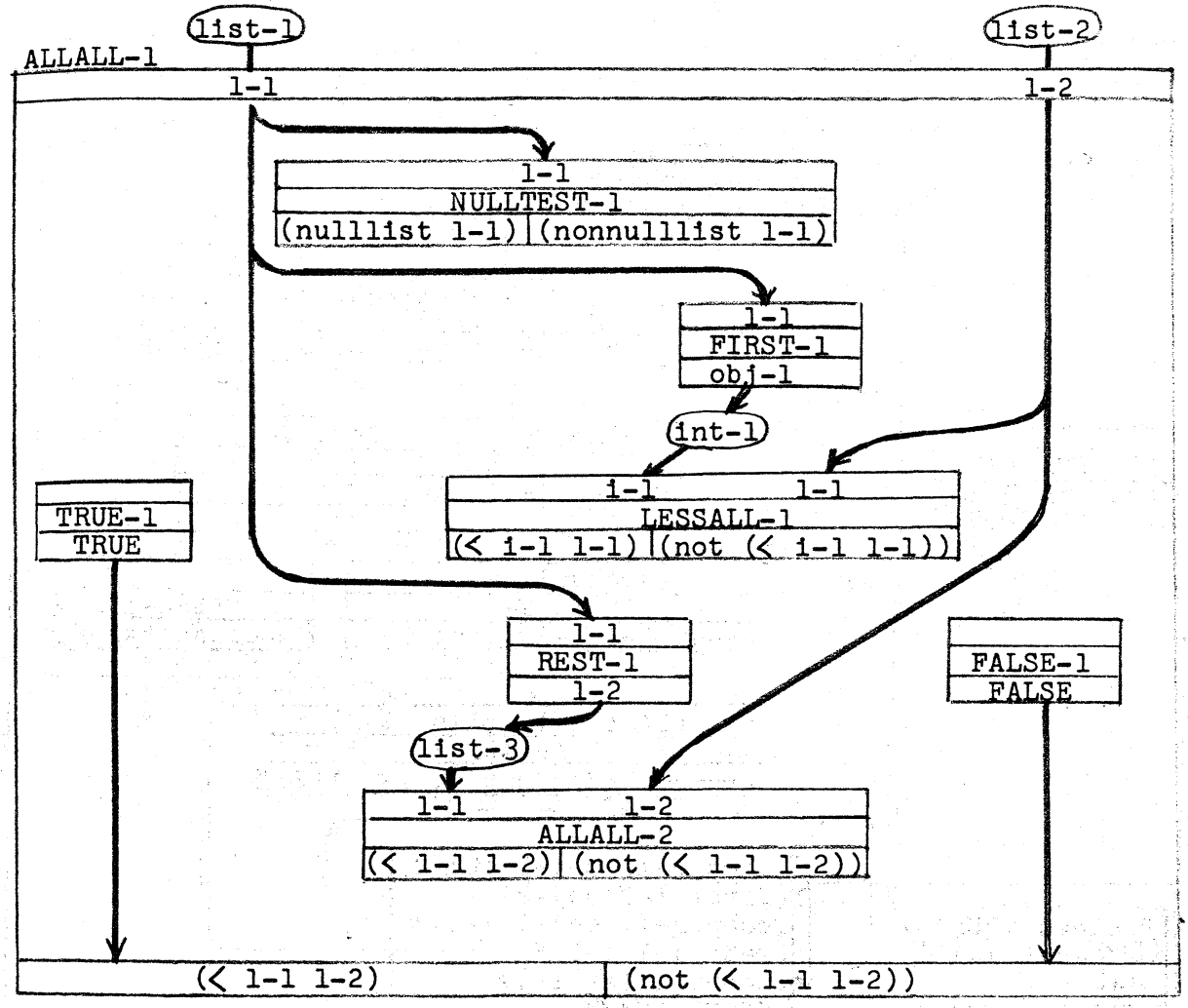


Fig.4(a) Dataflow of ALLALL Plan

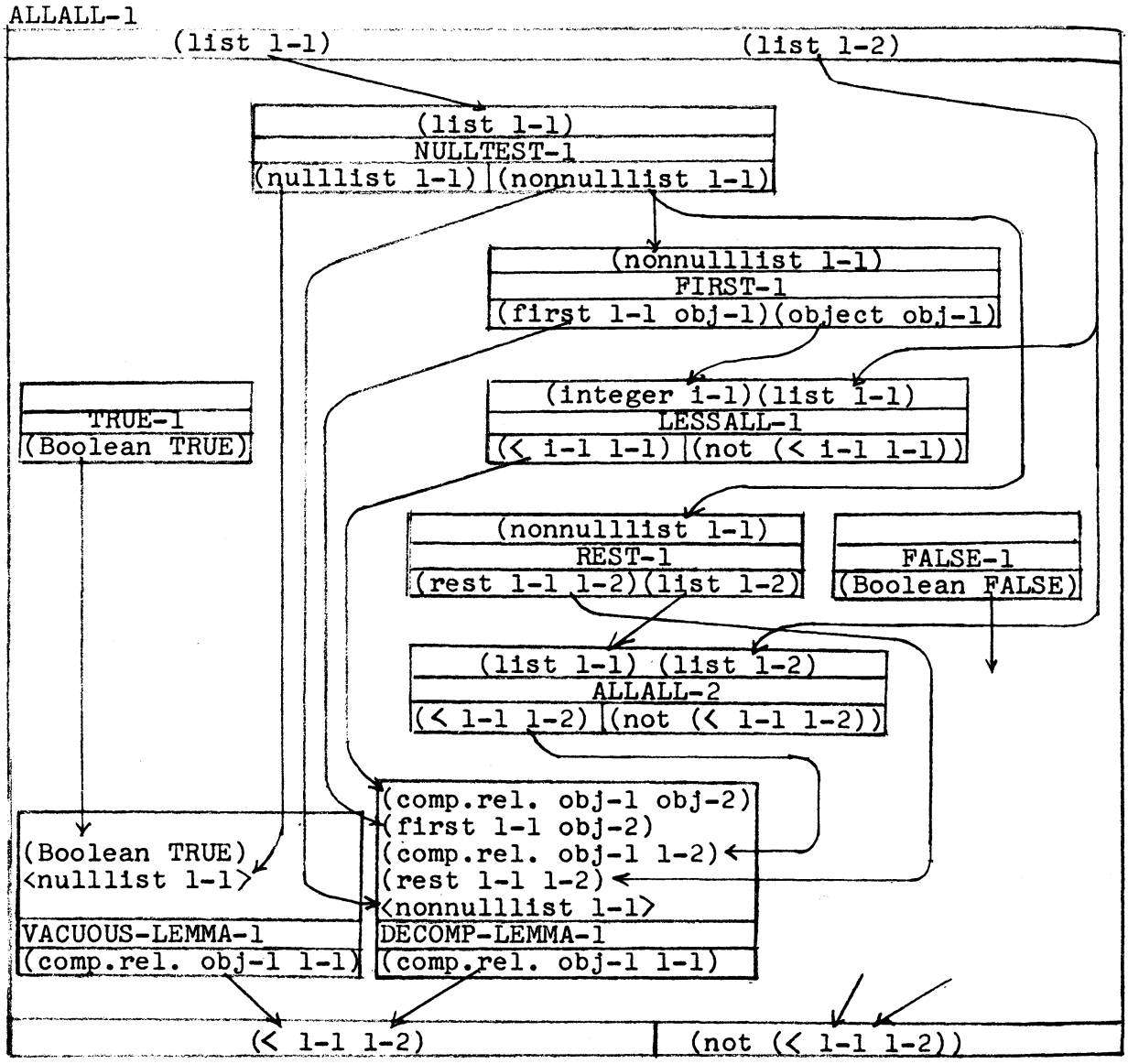


Fig.4(b) Logical Links of ALLALL Plan

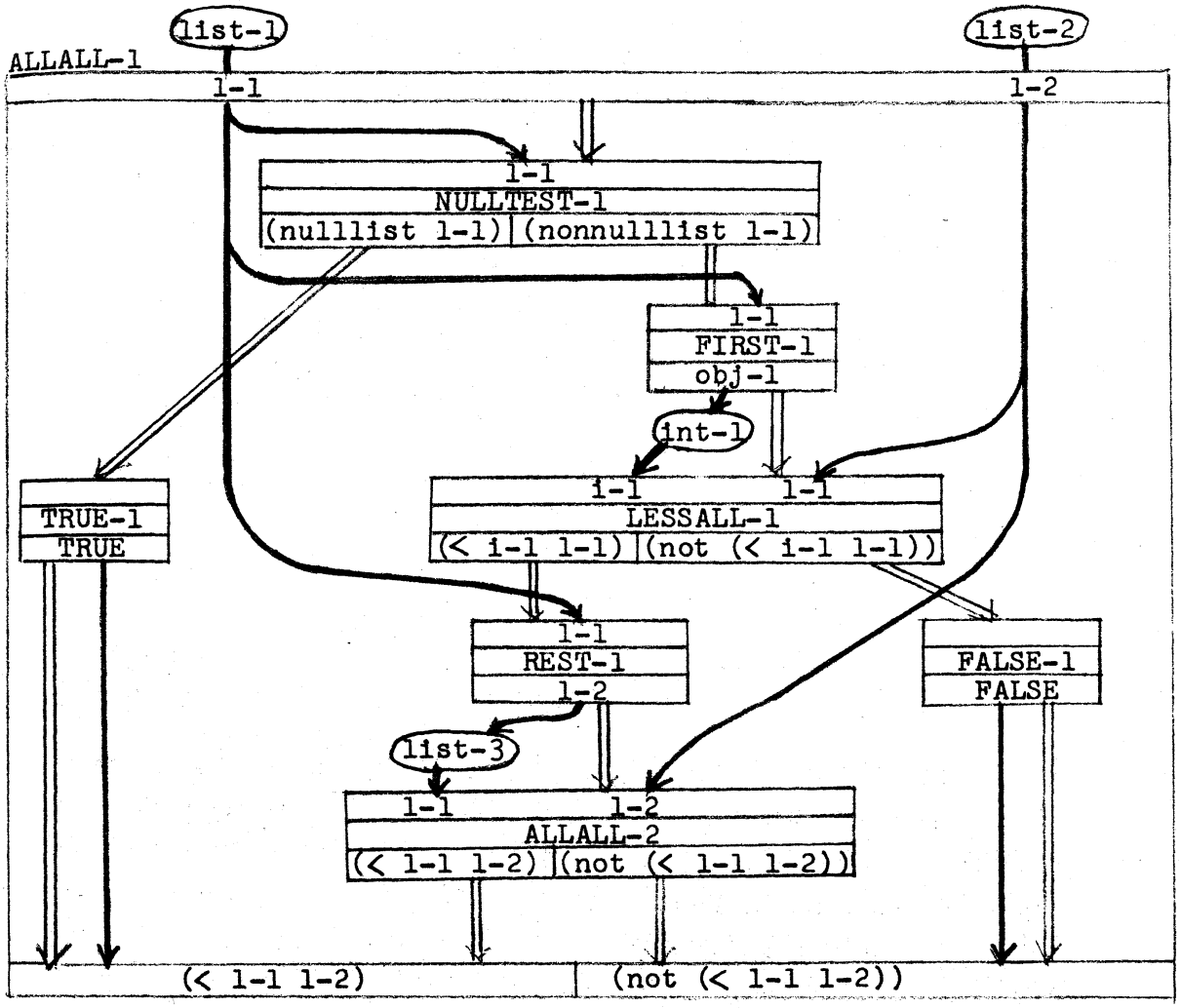


Fig.4(c) Data and Control Flow of ALLALL Plan

dataflow ➞ control-flow ⟹